04/11/2023
Saturday

## * Delete Node -

Cases - (1) Empty linked list.
(2) Single element in linked list.
(3) Delete head node.
(4) Delete tail node.
(5) Delete node in b/w linked list.

### Case 1 - Empty linked list -

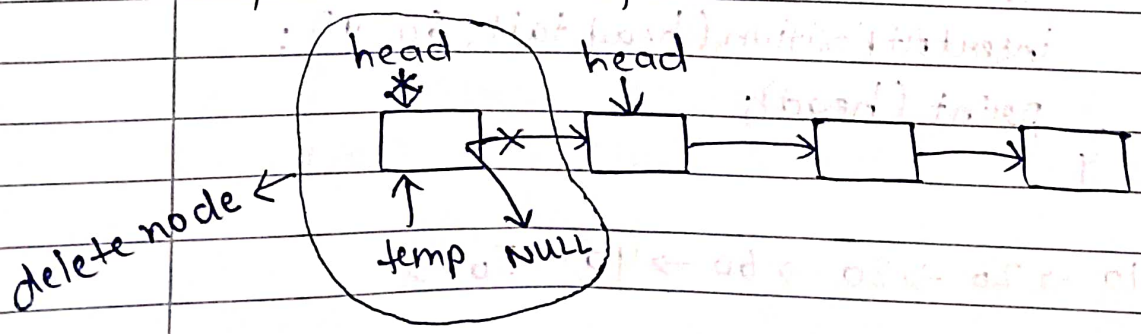
Nothing to delete. Simply, return.

### Case 2 - Single element in LL -

Step 1: Node* temp = head.
Step 2: delete temp.
Step 3: head = NULL, tail = NULL.

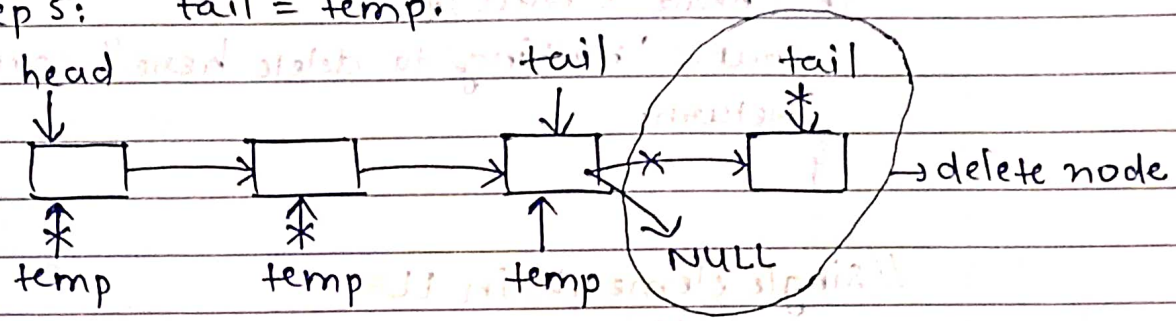
### Case 3 - Delete head node -

Step 1: Node* temp = head.
Step 2: head = temp -> next
Step 3: temp -> next = NULL.
Step 4: delete temp.



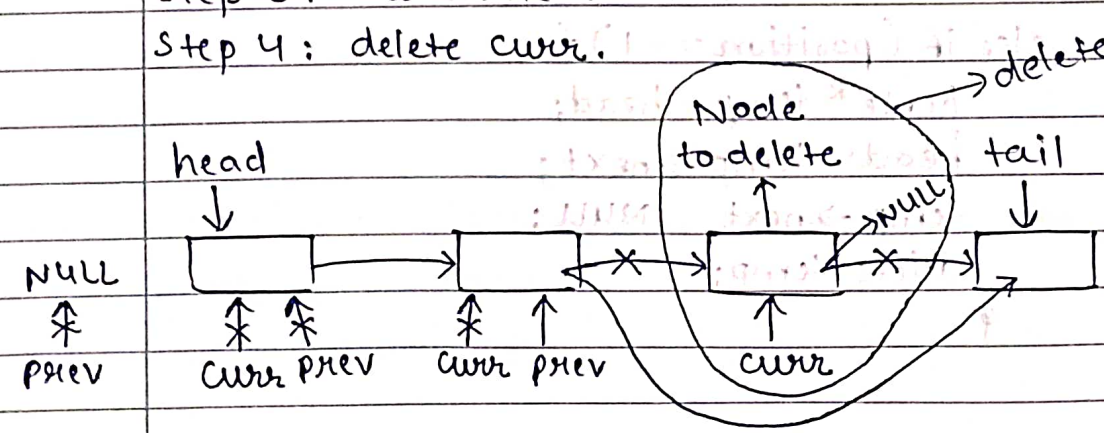
## Case 4 - Delete tail node -

Step 1: Node* temp = head.
Step 2: Traverse temp upto second last node.
Step 3: temp -> next = NULL.
Step 4: delete tail.
Step 5: tail = temp.




## Case 5 - Delete node in b/w LL -

Step 1: Traverse prev & curr upto required position.
Step 2: prev -> next = curr -> next.
Step 3: curr -> next = NULL.
Step 4: delete curr.




**Note -** Different cases isliye bn rahe h kyuki delete head ke case mai head change hoga, delete tail ke case mai tail change hoga and in b/w ke case mai head and tail remains unchanged.

Code →

```
void deleteNode(Node* &head, Node* &tail, int position){

    int length = lengthLL(head);

    //empty LL
    if (head == NULL){
        cout<< "nothing to delete here" <<endl;
        return;
    }

    //single element in LL
    else if (head == tail){
        Node* temp = head;
        delete temp;
        head = NULL;
        tail = NULL;
    }

    // delete head node
    else if (position == 1){
        Node* temp = head;
        head = temp->next;
        temp->next = NULL;
        delete temp;
    }
```

// delete tail node

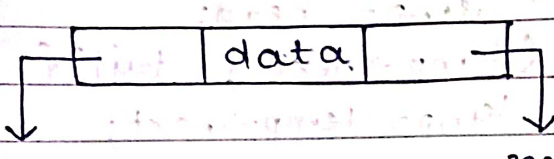
```
else if (position == length){
    Node* temp = head;
    while (temp->next != tail){
        temp = temp->next;
    }
    temp->next = NULL;
    delete tail;
    tail = temp;
}
```

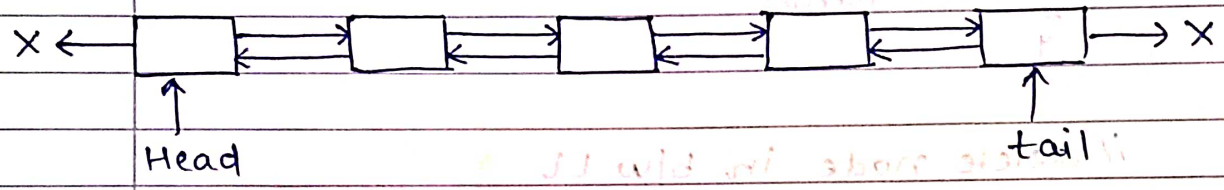
// delete node in b/w LL

```
else {
    Node* prev = NULL;
    Node* curr = head;
    while (position != 1){
        position--;
        prev = curr;
        curr = curr->next;
    }
    prev->next = curr->next;
    curr->next = NULL;
    delete curr;
}
}
```

* **Doubly - linked list -**

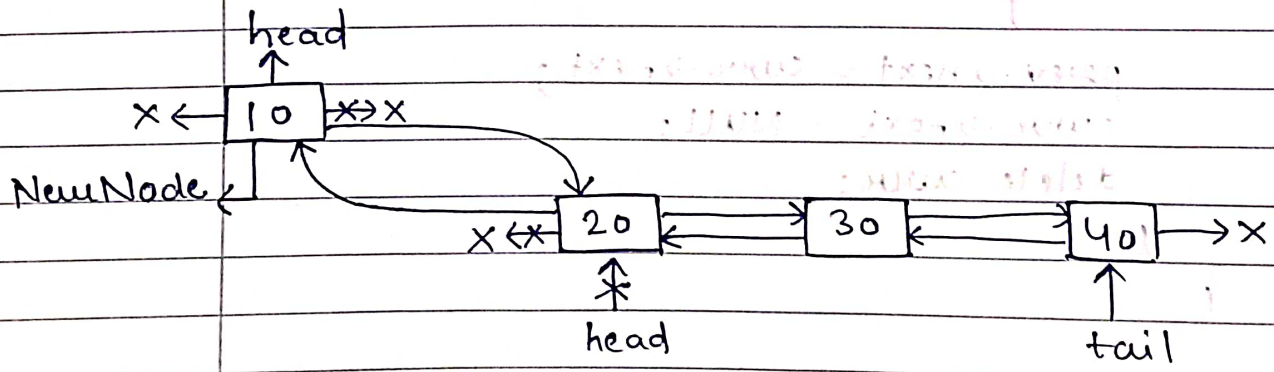
**Node -**




- **Insert at head in doubly linked list →**

Step 1: Create node.
Step 2: head → prev = NewNode.
Step 3: NewNode → next = head.
Step 4: head = NewNode.



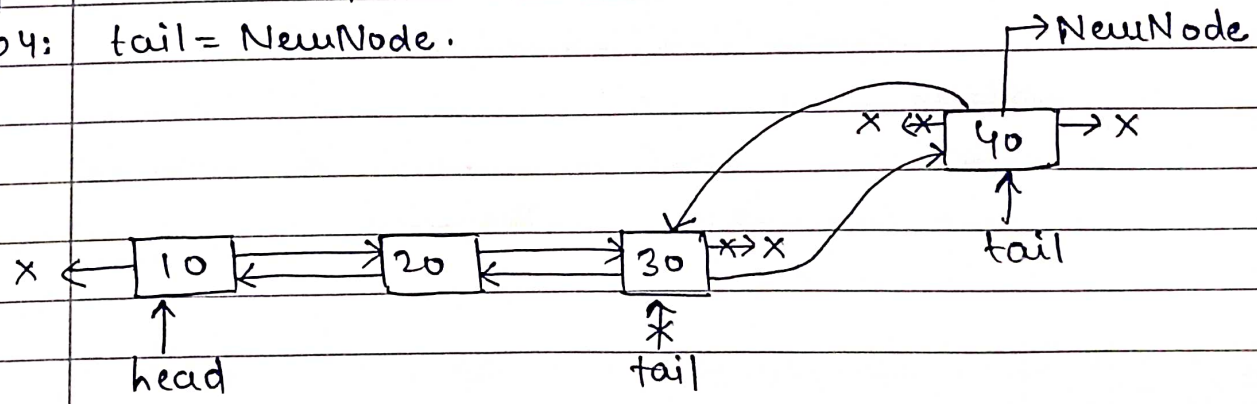

LL Before = 20 → 30 → 40
LL after = 10 → 20 → 30 → 40

Code -

```
void insertAtHead(Node* &head, Node* &tail, int data)
{
    if (head == NULL) {
        // Empty LL
        Node * NewNode = new Node(data);
        head = NewNode;
        tail = NewNode;
    }
    else {
        // non-empty LL
        Node * NewNode = new Node(data);
        head -> prev = NewNode;
        NewNode -> next = head;
        head = NewNode;
    }
}
```

- **Insert at tail in doubly LL →**

Step 1: Create node.
Step 2: tail -> next = NewNode.
Step 3: NewNode -> prev = tail.
Step 4: tail = NewNode.




LL Before = 10 -> 20 -> 30
LL after = 10 -> 20 -> 30 -> 40

Code -

```
void insertAtTail (Node* &head, Node* &tail, int data)
{
    if (tail == NULL) {
        // Empty LL
    Node* NewNode = new Node (data);
    head = NewNode;
    tail = NewNode;
    }
    else {
        // non-empty LL
    Node* NewNode = new Node (data);
    tail -> next = NewNode;
    NewNode -> prev = tail;
    tail = NewNode;
    }
}
```

- **Insert in b/w doubly LL -**

Step 1: Create node.

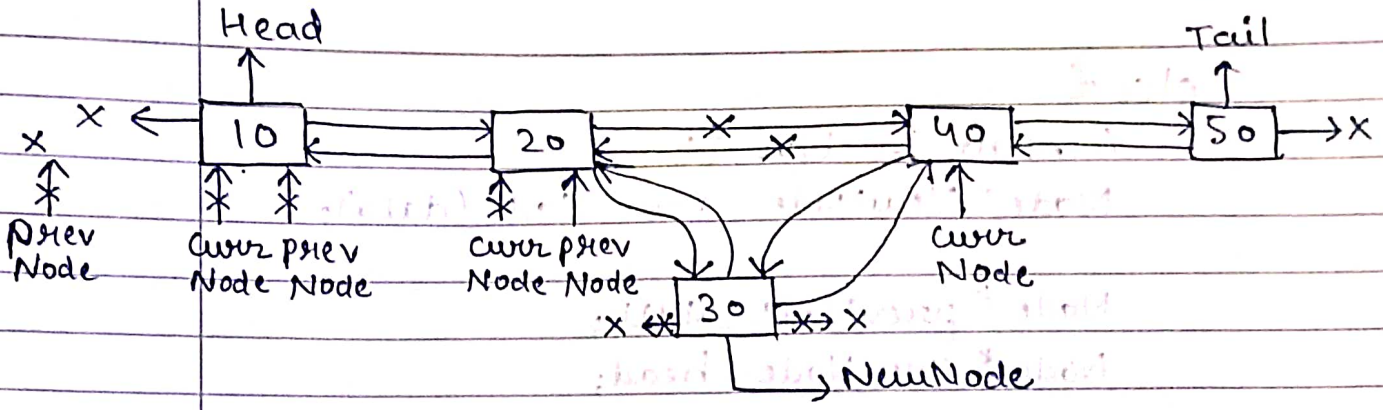
Step 2: Traverse prevNode and currNode upto the required positions.

Step 3: prevNode -> next = NewNode.

Step 4: NewNode -> prev = prevNode.

Step 5: NewNode -> next = currNode.

Step 6: currNode -> prev = NewNode.




Code →

```
void insertAtPosition (Node* &head, Node* &tail,
                       int data, int position)
{
  if (head == NULL){
    //Empty LL
    Node * NewNode = new Node (data);
    head = NewNode;
    tail = NewNode;
  }
  else {
    //non-empty LL
  int length= lengthLL (head);

  if (position == 1) {
    //insert at head
    insertAtHead (head, tail, data);
  }
  else if (position == length +1) {
    //insert at tail
    insertAtTail (head, tail, data);
  }
```

```
else {
    //insert in b/w LL ;
    Node * NewNode = new Node (data);

    Node * prevNode = NULL;
    Node * currNode = head;

    while (position != 1) {
            position --;
            prevNode = currNode;
            currNode = currNode -> next;
          }

    prevNode -> next = NewNode;
    NewNode -> prev = prevNode;
    NewNode -> next = currNode;
    currNode -> prev = NewNode;
    }
  }
}
```

- Delete node in doubly linked list -

Cases - (1) Empty LL
(2) Single element in LL
(3) Delete head node
(4) Delete tail node
(5) Delete node in b/w LL

Case 1 - Empty LL -

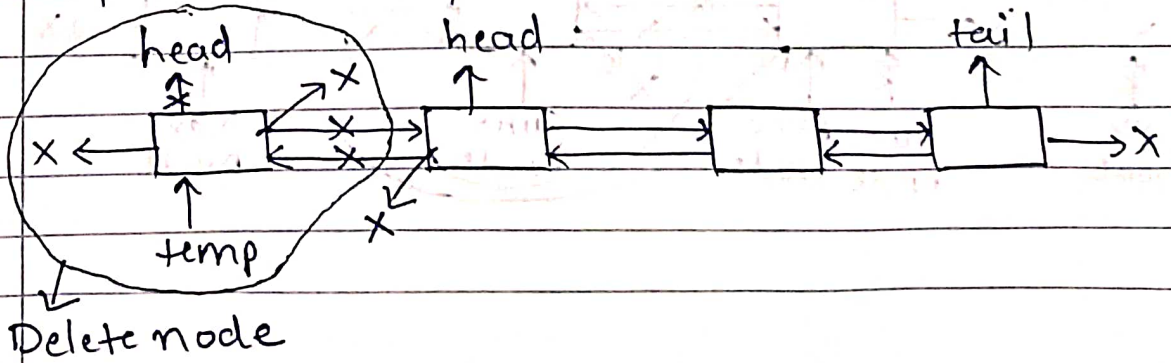
Nothing to delete. Simply, return.

Case 2 - Single element in LL -

Step 1 : Node* temp = head;
Step 2 : delete temp.
Step 3 : head = NULL, tail = NULL.

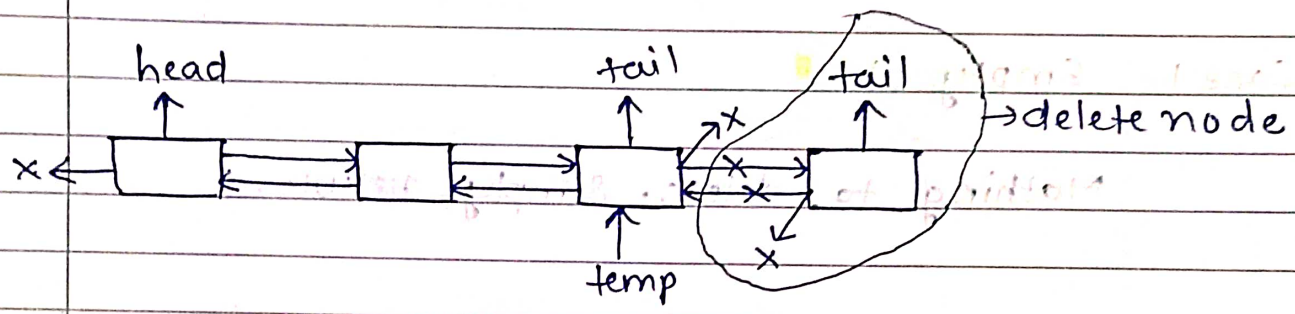
Case 3 - Delete head node -

Step 1: Node* temp = head.
Step 2: head = head -> next.
Step 3: temp -> next = NULL.
Step 4: head -> prev = NULL.
Step 5: delete temp.



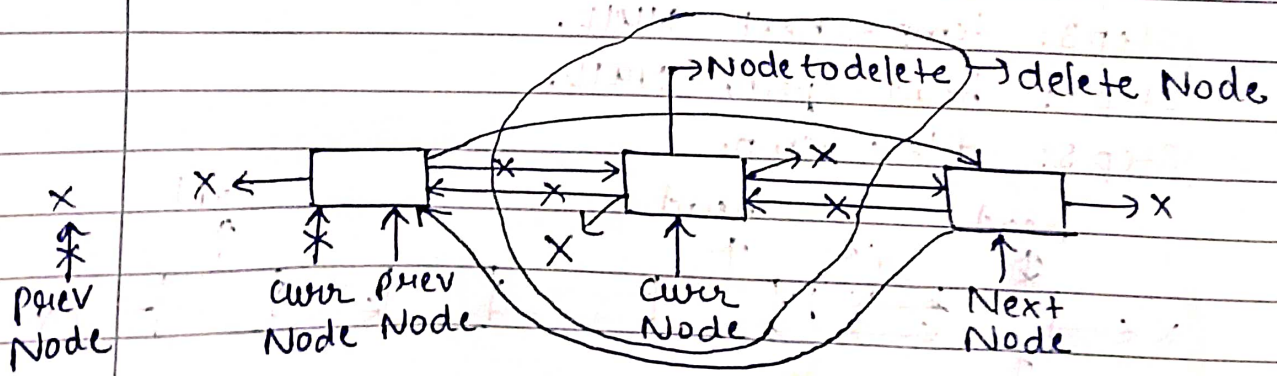
## Case 4:- Delete tail node -

Step 1: Node * temp = tail -> prev.
Step 2: temp -> next = NULL.
Step 3: tail -> prev = NULL.
Step 4: delete tail.
Step 5: tail = temp.




## Case 5 - Delete node in b/w LL -

Step 1: Traverise prev & curr upto required position.
Step 2: Node * NextNode = currNode -> next.
Step 3: prevNode -> next = NextNode.
Step 4: NextNode -> prev = prevNode.
Step 5: currNode -> prev = NULL.
Step 6: currNode -> next = NULL.
Step 7: delete currNode.

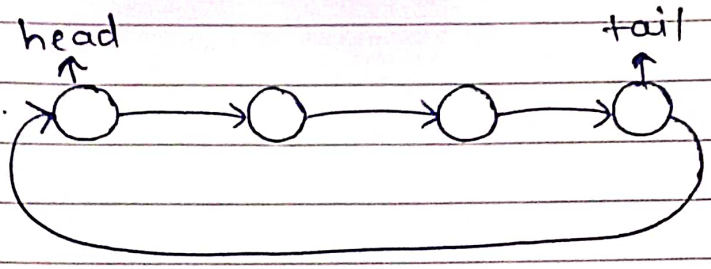


Code-

```
void deleteNode (Node* &head, Node* &tail,
                 int position)
{
  int length = lengthLL (head);

  if (head == NULL){
     //Empty LL
    cout << "nothing to delete" << endl;
     return;
  }

 else if (head == tail){
     //single element in LL
     Node* temp= head;
      delete temp;
      head= NULL;
      tail = NULL;
      return;
    }

 else if (position == 1){
     //delete head node
    Node* temp = head;
    head = head->next;
    temp->next = NULL;
    head->prev = NULL;
    delete temp;
 }
```

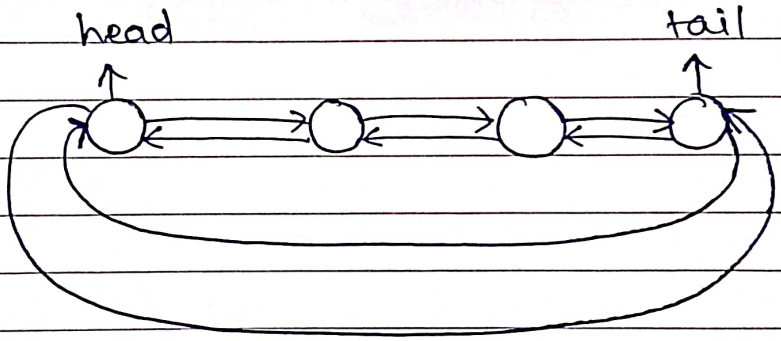
```
else if (position == length){
    //delete tail node
    Node * temp= tail ->prev;
    temp ->next = NULL;
    tail ->prev =NULL;
    delete tail;
    tail = temp;
}

else {
    //delete in b/w LL

    Node* prevNode = NULL;
    Node * currNode = head;
    while (position != 1){
        position--;
        prevNode = currNode;
        currNode = currNode ->next;
    }
    Node * NextNode = currNode ->next;

    prevNode ->next = NextNode;
    NextNode ->prev = prevNode;
    currNode ->prev =NULL;
    currNode ->next =NULL;
    delete currNode;
}
}
```

* Circular singly LL -




tail -> next = head

* Circular doubly LL -




tail -> next = head
head -> prev = tail